Различные реализации параллельных алгоритмов решения СЛАУ методом Гаусса

# Общие сведения

## Метод Гаусса

Метод Гаусса — это один из методов решения системы линейных алгебраических уравнений, метод последовательного исключения переменных, основанный на привидении системы уравнений к равносильной системе треугольного вида путем элементарных преобразований (домножение уравнения на ненулевой действительный коэффициент и прибавление к нему другого уравнения системы, домноженного на действительный коэффициент; перемена местами двух уравнений системы), из которой последовательно находятся все остальные переменные.

Пусть исходная система выглядит следующим образом:

## Прямой ход:

Прямой ход заключается в приведении системы к ступенчатой или треугольной форме путем элементарных преобразований.

В результате прямого хода получают систему вида:

## OpenMP

OpenMP - это открытый стандарт директив компилятора, библиотечных процедур и переменных окружения, которые предназначены разработки высокоуровневых многопоточных приложений на языках C/C++ и Fortran.

Для разработки многопоточных приложений необходимо выделять блоки кода, которые могут быть распределены между процессорами, на что уходит огромное количество времени. В большинстве программ, код, выполняемый на одном процессоре, зависит от результатов другого. Для упрощения разработки, программисты в 1980х создали специальные нотации, чтобы стандартизировать как будет проходить разбиение работы между отдельными процессорами и чтобы обеспечить правильный порядок обращений к данным. Нотация имеет вид специальных инструкций и директив, которые компилятор использует, чтобы создать конкретный код для выполнения на каждом процессоре.

OpenMP имеет ряд преимуществ, благодаря которым стоит обратить на него внимание:

* Он является наиболее широко-используемым стандартом для симметричных многопроцессорных систем
* Поддерживает три разных языка: Fortran, C, C++
* Довольно небольшая и простая спецификация
* Довольно много исследований проводится над ней, что позволяет держать ее в тренде с последними достижениями техники

# Реализация параллельных алгоритмов метода Гаусса на OpenMP

Полный код приложения, осуществляющего прямой ход метода Гаусса рассмотренными ниже способами приведен в приложении. Программа, к сожалению, не выполняет проверку на вырожденность матрицы, но гарантирует, что если решение СЛАУ существует, то прямой ход будет выполнен без ошибок.

Для решения берется случайно сгенерированная матрица и вектор ответа , вектор , сравнение результатов происходит с точностью , которая может появиться в виду операций над вещественными числами.

## Тайлинг: Линейный подход

Основным подходом при реализации многопоточных алгоритмов является тайлинг, рассмотрим линейную реализацию алгоритма прямого хода метода Гаусса:

for (int k = 0; k < N - 1; k++) {

for (int i = k + 1; i < N; i++) {

for (int j = k + 1; j < N + 1; j++) {

a[i][j] = a[i][j] - a[i][k] \* a[k][j] / a[k][k];

}

}

}

Основа алгоритма: проходимся по каждой строке, и отнимаем ее с соответствующим коэффициентом от всех последующих. Преобразуем алгоритм таким образом, чтобы указанные итерации осуществлялись последовательно по прямоугольным участкам, и обработку таких участков вынесем в отдельную функцию. Пусть – размеры этих участков, а – количество их в столбце и ряду соответственно.

for (int i\_gl = 0; i\_gl < Q1; ++i\_gl) {

for (int j\_gl = 0; j\_gl < Q2; ++j\_gl) {

// Начало тайла

for (int k = 0; k < N - 1; ++k) {

int start\_i = max(k+1, 1+i\_gl \* r1);

int finish\_i = min(1+r1\*(1 + i\_gl), N);

int start\_j = max(k+1, 1 + j\_gl \* r2);

int finish\_j = min(1+r2 \* (1 + j\_gl), N+1);

for (int i = start\_i; i < finish\_i; ++i) {

double coef = matrix[i][k] / matrix[k][k];

for (int j = start\_j; j < finish\_j; ++j) {

matrix[i][j] = matrix[i][j] - matrix[k][j] \* coef;

}

}

}

// Конец тайла

}

}

Таким образом, тайл можно представить следующим методом:

void tile(int i\_gl, int j\_gl, int r1, int r2, double\*\* matrix) {

for (int k = 0; k < N - 1; ++k) {

int start\_i = max(k+1, 1+i\_gl \* r1);

int finish\_i = min(1+r1\*(1 + i\_gl), N);

int start\_j = max(k+1, 1 + j\_gl \* r2);

int finish\_j = min(1+r2 \* (1 + j\_gl), N+1);

for (int i = start\_i; i < finish\_i; ++i) {

double coef = matrix[i][k] / matrix[k][k];

for (int j = start\_j; j < finish\_j; ++j) {

matrix[i][j] = matrix[i][j] - matrix[k][j] \* coef;

}

}

}

}

## Параллелизация внешнего и внутреннего цикла

При помощи директивы parallel можем распределить выполнение тайлов между процессорами двумя способами: распараллелив внешний цикл или внутренний.

Внешний:

#pragma omp parallel for

for (int i\_gl = 0; i\_gl < Q1; ++i\_gl) {

for (int j\_gl = 0; j\_gl < Q2; ++j\_gl) {

tile(k, i\_gl, j\_gl, r1, r2, a);

}

}

Внутренний:

for (int i\_gl = 0; i\_gl < Q1; ++i\_gl) {

#pragma omp parallel for

for (int j\_gl = 0; j\_gl < Q2; ++j\_gl) {

tile(i\_gl, j\_gl, r1, r2, a, N);

}

}

# Анализ быстродействия

## Преимущества параллельных подходов

## Изучение вырожденных случаев

### R = 1 (тайлы размером 1х1)

### R = N (один единственный тайл)

### R1 =1, R2 = N

### R1 = N, R2 = 1

## Изучение влияния размера тайла на быстродействие

## Изучение влияния числа процессов на быстродействие